train

November 30, 2020

1 Train

This file is used to train the networks. While training, we save a checkpoint file after each epoch of training which contains the model's parameters, the optimizer's parameters, the average loss over the epoch, and the average validation dice score. Using this approach, we can then later plot learning curves and restart training from any checkpoint.

[1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[2]:
```
!pip install pytorch-msssim
```

```
Requirement already satisfied: pytorch-msssim in /usr/local/lib/python3.6/dist-
packages (0.2.1)
Requirement already satisfied: torch in /usr/local/lib/python3.6/dist-packages
(from pytorch-msssim) (1.7.0+cu101)
Requirement already satisfied: future in /usr/local/lib/python3.6/dist-packages
(from torch->pytorch-msssim) (0.16.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages
(from torch->pytorch-msssim) (1.18.5)
Requirement already satisfied: typing-extensions in
/usr/local/lib/python3.6/dist-packages (from torch->pytorch-msssim) (3.7.4.3)
Requirement already satisfied: dataclasses in /usr/local/lib/python3.6/dist-
packages (from torch->pytorch-msssim) (0.8)
```

[3]:
```
# Import Modules
import sys
sys.path.insert(1, "/content/drive/My Drive/CMPUT511/Project/Code/RFDN")

import train
from RFDN import RFDN, RFDN1, RFDN2
from BaseN import BaseN
```

1.1 Training Loop

The cells below contain the training loop. The trainer object will take in the checkpoint file, data containing directory, learning rate, and model number. It will then train the network, checkpointing and saving in the same directory that contains the argument checkpoint file. Additionally, the trainer can load in the checkpoint file and resume training.

To train our networks, we use the mean L_1 loss, as specified in the RFDN paper:

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N} ||H(I_i^{LR}) - I_i^{HR}||_1$$

where H is the hypothesis of the model being trained, I_i^{LR} is the i^{th} pixel of the low-resolution image, I_i^{HR} is the i^{th} pixel of the high resolution imgae, θ are the model parameters, N is the number of pixels in the image, and $|| \cdot ||_1$ is the L_1 norm.

The creation and optimization of this loss function is taken care of by the Trainer class. Additionally, we use the Adam optimizer for training, which is also created and maintained by the Trainer class.

1.1.1 Preliminaries

In the next cell, we set up some preliminary variables.

[4]:
```
# Directory containing the data
data_dir = "/content/drive/My Drive/CMPUT511/Project/Data"

# Checkpoint file to save
checkpoint_file = "/content/drive/My Drive/CMPUT511/Project/Checkpoints/
 ↪checkpoint.tar"

# Number of training epochs
epochs = 1

# Whether or not to load the checkpoint file specified above
load = False

# Model number; used to train multiple models while naming checkpoint files␣
 ↪sequentially to ensure checkpoint files are not overwritten
model_num=0

# Learning rate and division constant
lr = 1e-4
div = 1.005

# Upscaling factor
upscale = 2
```

1.1.2 Model choice

In the cell below, we can choose which model to train. Simply uncomment the line of the model that should be trained, leaving all other lines commented.

```
[5]: model = RFDN1(nf=10, upscale=upscale)
     # model = RFDN(nf=10, upscale=upscale)
     # model = RFDN2(nf=10, upscale=upscale)
     # model = BaseN(nf=10, upscale=upscale)
     # model = FDCN(nf=10, upscale=upscale)
```

1.1.3 Train

In the following two cells, we create the Trainer object and begin training. The Trainer will attempt to minimize the mean L_1 loss using the Adam optimizer.

```
[6]: trainer = train.Trainer(model, checkpoint_file, data_dir, lr=lr, div=div,␣
      ↪num=model_num)
```

```
[7]: trainer.train(epochs, load=load)
```

```
100%|      | 800/800 [57:04<00:00,  4.28s/it]
```